NPS54-86-001PR

NAVAL POSTGRADUATE SCHOOL

Monterey, California

ADDITIONS TO THE KERMIT SYSTEM FOR

TRANSFERRING BINARY FILES THROUGH CMS

by

Norman R. Lyons

February 1986

Approved for public release. Distribution Unlimited.

Prepared for:
National Communications Systems
Arlington, VA 22204

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM. R. H. Shumaker
Superintendent

David A. Schrady
Provost

The research summarized herein was sponsored by National Communications System.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS 54-86-001PR	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 54
7a NAME OF MONITORING ORGANIZATION National Communications System	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b ADDRESS (City, State, and ZIP Code) 8th Street and South Courthouse Road Arlington, VA 22204
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NCS	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MIPR HC1001-6-10118	
8c. ADDRESS (City, State, and ZIP Code) 8th Street and South Courhouse Road Arlington, VA 22204	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

Additions to the Kermit System for Transferring Binary Files through CMS

12 PERSONAL AUTHOR(S)
Norman R. Lyons

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Project	FROM Jan 86 TO Feb 86	February 1986	42

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL Norman R. Lyons	22b TELEPHONE (Include Area Code) 408-646-2666
22c OFFICE SYMBOL 54Lb	

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

ADDITIONS TO THE KERMIT SYSTEM FOR TRANSFERRING BINARY FILES THROUGH CMS

by

Norman Lyons

Administrative Sciences Department
Naval Postgraduate School
Monterey, California 93943
February 1986

Summary

Problem

It is convenient to be able to store binary files on a mainframe computer. The mainframe computer can be used as a bulletin board that may be accessed by anyone owning a personal computer. Files may be left on this bulletin board and downloaded by anybody with legal access to the mainframe. Because of the design of the IBM mainframe, it has not been possible to download binary files. An IBM's communication processor strips off the eighth bit of each incoming byte. This renders binary files meaningless, and has precluded the use of IBM mainframes as a bulletin board.

Objective

The objective of this research is to develop a protocol whereby binary files can be transmitted to an IBM mainframe, left there on a bulletin board, and later downloaded to other microcomputers.

Approach

The approach taken in this research was to develop two C programs, PREPARE and UNPACK. These programs convert a binary file of eight bit words into one of six bit words that may be transmitted over a modem to an IBM mainframe. The UNPACK routine unpacks the prepared file once it has been downloaded from the mainframe. These files represent extensions to the KERMIT file transfer protocol.

Results and Conclusions

The PREPARE and UNPACK routines have been successfully tested and used on the IBM mainframe in conjunction with the KERMIT file transfer protocol.

CONTENTS

LIST OF FIGURES

Introduction

One of the factors contributing to the rise and widespread use of microcomputers has been computerized bulletin boards. A bulletin board is a resident program running in a computer equiped with a modem that can answer the telephone. Users of other microcomputers call this computer and hold a dialog with the bulletin board program. They can leave messages, or they can upload or download files.

It is this uploading and downloading of files that is one of the more useful features of a bulletin board. Users are able to store programs in both source and compiled modes. Compiled programs can be downloaded and used by other users, even though they do not have the compiler used in that language. Because bulletin boards generally can download files, all users may use the compiled code.

A whole culture has grown up of computer users who access and leave code on bulletin boards. In addition, they have developed a series of file transfer protocols and utility programs for bulletin boards. One example of such a program is the modem program PC-TALK III. This program is distributed as "freeware" or "shareware". This means that the program is left of bulletin boards and users who download it are encouraged to pass it around. A banner in the program asks for a donation of 35, but beyond that there is no formal means of distributing this program. Users are encouraged to use it and pass it around, and their conscience directs them as to whether or not they should pay the fee. The banner also suggests that users who use this in a corporate environment are required to pay the fee. But again, there is no means of enforcement.

This mode of transmission works better than one might expect. Programs found on bulletin boards are widely distributed and many are surprisingly good. The shareware approach is a strange approach to distributing software. On one hand, it allows software to be widely distributed without the usual marketing setup costs. However, it is not proved to be a very consistent way of generating income for the developers.

It remains a small and interesting subculture in microcomputing. In spite of the fact that software is free, it is of high technical quality. The PC-TALK III modem package has incorporated in it a protocol called

the XMODEM protocol. This protocol allows the uploading and downloading of binary files and is largely error free. This is the mechanism used by most microcomputer enthusiasts for transferring programs around the bulletin boards.

Another utility that is commonly used by bulletin board enthusiasts is a program called ARC for archiver. There are problems when distributing a large system via a bulletin board. A system is usually made up of several programs and documents. It can be quite tedious to have to upload or download each of the systems individually. One way to do that would be to concatenate the systems into a single file.

While one is doing this concatenation, it would also be helpful if the files were compressed. There is a great deal of redundant information in many data files. Such things as repeated comments, blank lines and so on could be compressed out by any of several data compression routines. The archive utility is one such program that will compress programs and then put them in a library archive. It also has the facilities for listing the contents of an archive and extracting files from the archive and adding new files to the archive.

In a short period of time, this archive utility has become quite popular with bulletin board enthusiasts. The latest version of this utility, ARC 4.5, is described in the document distributed with it that is an appendix to this report. Note that this utility if used in a government environment is not considered freeware. A payment of 35 to the developer is required.

When transferring files through a mainframe system, a different approach is required. The XMODEM protocol and PC-TALK are implemented as programs on micros. This means that if you have a microcomputer, you can use PC-TALK to access a mainframe, but the mainframe probably does not have the means to upload or download binary files.

For mainframe systems, a different program has come into widespread use. This is the KERMIT program. The KERMIT program is named after Kermit the Frog of Sesame Street fame. Presumably, it is supposed to be user friendly like the original Kermit. It does not particularly seem user friendly, however. It is a very low level modem program and as such, the user has to perform many tasks that are taken care of automatically on more sophisticated modem programs. For example, KERMIT does not support the autodial features common to many modems. The KERMIT program was developed at Columbia University with contributions made by people all over the country. The original copies of KERMIT and the documentation for KERMIT are maintained at Columbia and widely accessible over the ARPANET.

The KERMIT program is intended for use on networks. The networks may be large and sophisticated like the ARPANET. Or they may be very simple. You could have two computers communicating in a two node network linked together by cable on RS232 interfaces. To use KERMIT, the user must have access to both machines. Then on both machines, the user starts a copy of KERMIT. The user must be able to switch freely back and forth between the two machines.

He then sets up the negotiation for the file transfer between the two KERMITS, setting them to compatible file transfer settings. One this initial negotiation has been done, the user sets one KERMIT to send and the other to receive. From here on out, the file transfer takes place automatically. KERMIT's simplicity is a real advantage. It means that the KERMIT program can be implemented on a wide variety of machines. The list includes virtually all of the popular mainframe and microcomputer systems. If you have a computer, chances are there is a KERMIT package that is written for you.

There is a KERMIT package in use on the IBM mainframe. This KERMIT protocol works conventionally like other KERMITs. However on an IBM mainframe, there are additional problems. The communications processor used with an IBM mainframe does not accept eight bit bytes. It will strip off the eighth bit on a computer byte and only allow seven bits to the mainframe.

Besides this, there are additional problems with the IBM's handling of this data. Not all of the seven bit bytes are created equal. In transmitting data back to the user, if the IBM system detects the code for a carriage return (hexidecimal 0D). The system will manipulate this character, appending a line feed (hexadecimal 0A) to it. Thus it turns out that what you send to an IBM system is not necessarily what you will receive. As a result, the developers of KERMIT have never gotten around to creating a protocol that will allow them to send and receive eight bit files on an IBM mainframe.

That was the major purpose of this effort. We needed to develop a protocol that would allow us to send eight bit files through an IBM mainframe so that we could use that mainframe as a bulletin board. KERMIT offers may advantages for error free file transfer. One would only want to use this protocol for files that had to be transmitted without error. Text files could be sent by ASCII protocols. But KERMIT is essential for transfer of binary files to and from a mainframe.

This is the only way in which we can be sure that the transmission takes place error free. ASCII file transfers frequently involve missing characters or missing lines. To allow us to use KERMIT successfully, routines had to be developed that would pack and unpack a binary file

for transmission via the KERMIT protocol. This is the purpose of the routines described in the next section.

The PREPARE and UNPACK Routines

Two routines described here are called the PREPARE and the UNPACK routines. A complete listing of these routines may be found in the appendix. They are written in LATTICE C. The PREPARE routine prepares a file for transmission by KERMIT. The UNPACK routine takes a file generated by the PREPARE routine and converts it back to its original form.

The PREPARE routine takes two parameters. The first is the name of the input file and can be any legal DOS filename and file type. The second parameter is a file name only. By default, PREPARE will choose the file type XMT for the output file. If these files are successfully declared, PREPARE then begins its work. It first writes a banner line on the file. It writes the text "IBM mainframe XFER prepare file". This line serves to identify the file as a PREPARE file and keeps UNPACK from trying to process an ordinary file for output. Next, PREPARE writes the name of the input file. This allows the UNPACK routine to produce the file of the same name that is initially input.

The packing approach used by the PREPARE routine is quite simple. Three eight byte computer words are shifted so they are written as four six byte computer words, preserving the same 24 bits of data. Each of the four six bit words have a hexadecimal 20 added to them. This shifts the scale of the output numbers so that at no point is a hexadecimal 0D (carriage return) output.

This prevents the IBM from appending the hexadecimal 0A (line feed) to what it thinks is a carriage return. An example of the shift and recoding process in given in Figure 1. This means that a resulting output file is about 1/3 larger than the original input file. After all bytes and binary files have been processed, and end of file marker, a hexadecimal 64 is output. This notifies the UNPACK routine of the point of which there are no more characters prepared by the PREPARE routine. The routine then outputs a line telling the number of bytes processed and teminates.

The UNPACK routine reverses the work of the PREPARE routine. It first checks to see if the first line of the file is the phrase "IBM mainframe XFER prepare file". If the first line is not this phrase, the routine terminates. This check is put in place to keep the UNPACK

Input			Output		
Hex	Binary		Shift	Add 20 hex	Hex out
AB	10101011	--->	101010 + 100000 = 1001010 = 4A		
1A	00011010		110001 + 100000 = 1010001 = 51		
C1	11000001		101011 + 100000 = 1001011 = 4B		
			000001 + 100000 = 100001 = 21		

Figure 1: Transformations by the PREPARE Routine

routine from trying to process randomly selected binary files. If the input files passes this test, the routine then reads in the name of the output file. A file of that name is assigned, and the UNPACK routine starts processing the data in the file. The UNPACK routine takes only one parameter. The file name of the input file. There is no extension on this file. The extension is assumed to be the characters XMT.

The UNPACK routine reads in four words of six bit characters. The hexadecimal 20 is subtracted from each of these characters and then they are packed into three words of eight bit characters. This reverses the action of the PREPARE routine. The process outlined in Figure 2.

This completes the discussion of the PREPARE and UNPACK routines. The interested reader is refered to the appendices with the listings of the C program. This shows the complete workings of the algorithm for packing and unpacking the binary files. If these routines are used in conjunction with the ARC utility mentioned earlier, the number of bits that have to be transferred is roughly equal to the original binary file size. The ARC utility compresses a binary file by about 1/3. The PREPARE utility then expands the size of the compressed file by about 1/3 so there is no net change in the process. The files that are transmitted to the mainframe computer are not excutable. After being processed by the PREPARE and ARC routines back on a microcomputer, they are again excutable programs. The next session discusses a sample dialog with these routines.

Input					Output	
Hex	Binary	Subtract 20 hex			Shift and Pack	Hex out
4A	= 1001010	- 100000	= 101010	--->	10101011	AB
51	= 1010001	- 100000	= 110001		00011010	1A
4B	= 1001011	- 100000	= 101011		11000001	C1
21	= 100001	- 100000	= 000001			

Figure 2: Transformations by the UNPACK Routine

File Transfer Example

This section shows a sample file transfer. First we upload to a host computer. After this we download that to a microcomputer from that same host computer. Figure 3 shows the complete dialog. At line number 1, we used the ARC routine to use the archive the two files NCS.EXE, and NCS.BAS. The result will be a new file called NCS.ARC. This new file is an Archive file contained of a compressed version of the two files in the archives. Refer to the documentation in the appendix for information on how to use the ARC utility.

After we have successfully created the archive, we use the PREPARE utility to generate a transmittal file. This is shown in line number 2. We call PREPARE on the file NCS.ARC to create the file NCS.XMT. At line number 4, we do a directory to show the files that we have created. At this point, we are ready to initiate the KERMIT processor. This is done in line number 5 by typing MSKERMIT.

KERMIT has a simple command set. Refer to a KERMIT documentation for more details. One can get help for KERMIT by typing a question mark, in line 6. This gives the names of the KERMIT commands. On line 7, we begin the setup of our KERMIT on the microcomputer. We must set the baud rate to 1200. We must perform a routine to tell the microcomputer we will be talking to an IBM machine. Finally we connect to the mainframe. This is done in line 7, 8 and 9. KERMIT prints out a messsage to notifies us that we are ready to begin connection to the host.

At line 10, we type in the autodial sequence for the Hayes modem. This is the string ATDT6462709 in line number 10. Be sure to type these characters in upper case. The characters come out doubled because we have changed duplex in the DO IBM routine. KERMIT then notifies us that we have connected to the machine. We get the characteristic banner headline and then do a carriage return in line 11. At this point, we can begin our log on sequence. Log in with the account number and then the password when the system asks for it in lines 12 and 13.

We get the message from the computer and then are ready to begin initiation of CMS. This is done in line 14 by typing a carriage return. The begin the normal log on for CMS, executing any files in our

```
C>arc a ncs ncs.exe ncs.bas          <-------------- 1.
Creating new archive: NCS.ARC
Adding file:   NCS.BAS        analyzing, crunching, done.
Adding file:   NCS.EXE        analyzing, crunching, done.


C>prepare ncs.arc ncs                <-------------- 2.

Bytes input:   11405
Bytes output:  15207


C>dir ncs.*                          <-------------- 4.

 Volume in drive C is C-DRIVE
 Directory of  C:\MODEMS

NCS      EXE     9457   1-11-86   7:00p
NCS      BAS     9273   1-11-86   6:54p
NCS      XMT    15250   1-18-86  11:37a
NCS      ARC    11405   1-11-86   7:00p
        4 File(s)   1499136 bytes free


C>mskermit                           <-------------- 5.
IBM-PC Kermit-MS V2.27
Type ? for help

Kermit-MS>?                          <-------------- 6.
BYE            CLOSE          CONNECT        DEFINE
DELETE         DIRECTORY      DO             EXIT
FINISH         GET            HELP           LOCAL
LOG            LOGOUT         PUSH           QUIT
RECEIVE        REMOTE         RUN            SEND
SERVER         SET            SHOW           SPACE
STATUS         TAKE
Kermit-MS>set baud 1200              <-------------- 7.
Kermit-MS>do ibm                     <-------------- 8.
Kermit-MS>connect                    <-------------- 9.

[Connecting to host, type Control-] C to return to PC]


AATTDDTT66446622770099               <-------------- 10.
CONNECT

VM/370 ONLINE
```

```
|                                           <-------------- 11.

.l 1234p                                    <-------------- 12.
ENTER PASSWORD:
.passwordSSSSSSSSSSSSSSSSSS                     <-------------- 13.
LOGMSG - 13:50:49 PST FRIDAY 01/17/86
*************************************************************************
*****  .  SEE "NEWS" FOR WEEKEND HOLIDAY SCHEDULE.                  *****
*************************************************************************
LOGON AT 11:46:40 PST SATURDAY 01/18/86
 CMS print files are now printed (by default)
 on the NPGBATCH printer.
CMS 3.1
.                                           <-------------- 14.
CP TERM ESCAPE OFF CHARDEL OFF LINEDEL OFF
EXEC FIXBKSP
FIXING BACKSPACE FOR HOME COMPUTER ACCESS
GLOBAL TXTLIB VFORTLIB CMSLIB FORTMOD2 MOD2EEH IMSLSP NONIMSL
GLOBAL LOADLIB VFLODLIB
R; T=0.01/0.05 11:48:00
.linkto kermit                              <-------------- 15.
B (120) R/O
Linked to user KERMIT disk 191 as 120 (B).
R; T=0.01/0.04 11:49:17
.kermit                                     <-------------- 16.
Kermit CMS Version 2.01
Enter ? for a list of valid commands

KERMIT-CMS>.receive                           <-------------- 17.
Ctrl-]C                                     <-------------- 18.
Kermit-MS>send ncs.xmt                        <-------------- 19.


            File name: NCS.XMT
  KBytes transferred: 2
 Percent transferred: 16%
              Sending: In progress

   Number of packets: 30
   Number of retries: 0
          Last error: None
        Last warning: None
```

```
Kermit-MS>connect                          <-------------- 20.

q                                          <-------------- 21
R; T=0.22/1.75 11:57:05
.logout                                    <-------------- 22.
CONNECT= 00:10:34 VIRTCPU= 000:00.26 TOTCPU= 000:02.00
CONNECT=   0.88 TOTCPU=   0.33 SIO=   0.07  TOTAL=    1.28
MULT BY SHIFT FACTOR: =1(DAY), =0.25(EVE), =0.10(NIGHTS)
LOGOFF AT 11:57:14 PST SATURDAY 01/18/86
xp°
NO CARRIER
Ctrl-]C                                    <-------------- 23.
Kermit-MS>q                                <-------------- 24.
```

Figure 3:   Transmitting a File Using ARC, PREPARE and KERMIT

PROFILE EXEC (if any). At line l5, we link to the virtual machine containing KERMIT. This gives access to the KERMIT routines on the IBM. At line 16 we initiate the KERMIT processor on the IBM system. Once again we get the standard KERMIT input prompt.

We now have the two KERMIT's in the two computers ready to go. We are going to be sending a file from the microcomputer to the mainframe so our next command tells the mainframe KERMIT to receive. This is done in line number 17. Now we must go back to the microcomputer and initiate sending process. We do this by typing the sequence Ctrl-]C (hold down the Ctrl key and hit the "]" and "C" keys in succession). This sends us back to the microcomputer. This is done in line number 18. Once we are back in the microcomputer, we can initiate the sending of the file. This is done in line number 19 by typing the sequence SEND NCS.XMT.

KERMIT clears the screen and displays a message showing the status of the file transfer. percent of the entire filed that is transferred. At this point, the user is talking to the microcomputer KERMIT. When the sending terminates, a status message say the sending is completed and the microcomputer KERMIT prompt will appear. To get back to the mainframe, you must issue a connect command. This is done in line number 20. The screen again clears and the status of the mainframe is displayed on the machine. You are talking now to the mainframe KERMIT. To get back off the mainframe you must issue a quit command. This is done in line number 21. After that, you log out from

the mainframe in line number 22. Once you have logged off the mainframe, the carrier is dropped.

At this point you are talking to a dead mainframe and you must get back to the KERMIT in your microcomputer. This is done in line number 23 by typing the Ctrl-]C again. You are now back at your microcomputer KERMIT. Exit from it by typing a Q in line number 24.

Figure 4 gives an example of the use of KERMIT, UNPACK, and ARC to download a file from the mainframe. We begin on the microcomputer by initializing KERMIT. Connection to KERMIT is accomplished by typing MSKERMIT in line number 1. Once we are in KERMIT we must set the communications parameters. This is done by typing SET BAUD 1200, DO IBM, and CONNECT in line numbers 2, 3 and 4. KERMIT responds by connecting to the host. Once we are connected to the host we must type in the autodial sequence. We shift to caps on and then type ATDT6462709. Again, KERMIT doubles the characters because we are connected in half duplex mode.

KERMIT connects, and we get the IBM 370 prompt. We respond with the usual login sequence in line numbers 8 through 10. In line number 11 we link to KERMIT. This links us to the virtual machine that contains the KERMIT routines. In line number 12, we initiate KERMIT on the mainframe. Now, we are ready to begin transmitting our file down to the microcomputer. This time we enter the command line SEND NCS XMT in line number 13. This initiates the sending.

Now, we must go to the microcomputer and tell it to receive. We do the usual Ctrl-]C to transfer control to the microcomputer's KERMIT. Now, the prompt in line number 15 indicates that we are speaking to the microcomputer. At this point, we type RECEIVE. The screen clears, and we begin to see the status information on the transfer. When thE transfer is completed, the prompt line for the microcomputer KERMIT comes up again. In line number 16, we type CONNECT, and this connects us back to the mainframe KERMIT.

Once we are connected to the mainframe KERMIT, we quit KERMIT, and then log out of the machine in line numbers 17 and 18. Once again, we are talking to a dead computer since the mainframe has dropped the carrier. In line number 19, we type the Ctrl-]C sequence again, and this connects us back to the microcomputer KERMIT. In line number 20, we type a Q and we are ready to begin work on the microcomputer. We have received the file NCS.XMT properly, and now we can begin to unpack it.

We do this in line number 21 by typing a line number UNPACK NCS. The UNPACK routine responds by giving you the name of the

```
C>mskermit                              <-------------- 1.
IBM-PC Kermit-MS V2.27
Type ? for help

Kermit-MS>set baud 1200                 <-------------- 2.
Kermit-MS>do ibm                        <-------------- 3.
Kermit-MS>connect                       <-------------- 4.

[Connecting to host, type Control-] C to return to PC]


AATTDDTT66446622770099                  <-------------- 5.

CONNECT                                 <-------------- 6.
VM/370 ONLINE
|                                       <-------------- 7.

.l 1234p                                <-------------- 8.
ENTER PASSWORD:
.passwordSSSSSSSSSSSSSSSSS              <-------------- 9.
LOGMSG - 13:50:49 PST FRIDAY 01/17/86
*******************************************************************
*****     SEE "NEWS" FOR WEEKEND HOLIDAY SCHEDULE.           *****
*******************************************************************
LOGON AT 12:06:52 PST SATURDAY 01/18/86
 CMS print files are now printed (by default)
 on the NPGBATCH printer.
CMS 3.1
.                                       <-------------- 10.
CP TERM ESCAPE OFF CHARDEL OFF LINEDEL OFF
EXEC FIXBKSP
FIXING BACKSPACE FOR HOME COMPUTER ACCESS
GLOBAL TXTLIB VFORTLIB CMSLIB FORTMOD2 MOD2EEH IMSLSP NONIMSL
GLOBAL LOADLIB VFLODLIB
R; T=0.01/0.06 12:07:47
.linkto kermit                          <-------------- 11.
B (120) R/O
Linked to user KERMIT disk 191 as 120 (B).
R; T=0.01/0.04 12:08:42
.kermit                                 <-------------- 12.
Kermit CMS Version 2.01
Enter ? for a list of valid commands

Kermit-CMS>.send ncs xmt                <-------------- 13.
Ctrl]-C                                 <-------------- 14.
Kermit-MS>receive                       <-------------- 15.
```

```
          File name: NCS.XMT
  KBytes transferred: 1

          Receiving: In progress

   Number of packets: 25
   Number of retries: 1
         Last error: None
       Last warning: None




Kermit-MS>connect                     <-------------- 16.

q                                    <-------------- 17.
R; T=0.25/1.75 12:16:04
.logout                              <-------------- 18.
CONNECT= 00:09:16 VIRTCPU= 000:00.28 TOTCPU= 000:02.00
CONNECT=   0.77 TOTCPU=   0.33 SIO=   0.05  TOTAL=    1.15
MULT BY SHIFT FACTOR: =1(DAY), =0.25(EVE), =0.10(NIGHTS)
LOGOFF AT 12:16:09 PST SATURDAY 01/18/86
f6m
NO CARRIER
Ctrl-]C                              <-------------- 19.
Kermit-MS>q                          <-------------- 20.

C>unpack ncs                          <-------------- 21.


The output file name will be:

     NCS.ARC

Is this acceptable to you?
1 = Yes, anything else = No
1                                    <-------------- 22.

Bytes input:   15207
Bytes output:  11405


C>arc x ncs                          <-------------- 23.
```

```
Extracting file: NCS.BAS
Extracting file: NCS.EXE

C>dir ncs.*                                  <-------------- 24.

 Volume in drive C is C-DRIVE
 Directory of  C:\MODEMS

NCS      XMT     15252    1-18-86  12:15p
NCS      ARC     11405    1-18-86  12:21p
NCS      BAS      9273    1-11-86   6:54p
NCS      EXE      9457    1-11-86   7:00p
         4 File(s)    1441792 bytes free
```

Figure 4: Receiving a File Using KERMIT and UNPACK

file it is going to write, in this case, NCS.ARC and asks if this is this acceptable to you. It does this to prevent you from overwriting any files you may have of the same name. If this name is acceptable, type 1 and then continue. The UNPACK routine responds by telling you the number of bytes it has processed.

Now, we have a new file called NCS.ARC. We can now use the ARC routine arch to extract the files in the archives. So in line number 23, we type ARC X NCS to extract the files from the archives. The ARC routine comes back and gives us the name of the files it is extracting. Then in line number 24, we do a directory to show the files have been extracted successfully.

Appendix A

PREPARE Routine Listing

```
#include <stdio.h>
#define FNL 80   /* maximum file name length */

main(argc,argv)
   int argc;
   char *argv[];

/* Author:  Norman R. Lyons
             Naval Postgraduate School

   Date:    January 1986

   Version: 1.0

   Calling Sequence:  PREPARE infile[.ext] outfile

   Output:  outfile.XMT

   The PREPARE routine takes a file of eight bit bytes and
   converts it into one of six bit bytes that are ASCII
   characters on a range 32-95.  This file may be transmitted to
   and stored on an IBM mainframe using the KERMIT modem package.
   This allows storage of binary (eight bit bytes) files on
   IBM systems.  */

{
   FILE *infile, *outfile, *fopen();
   char filename[FNL],inname[FNL];
   int c,i,outc,rest;
   long int countin,countout;
   strcpy(filename,argv[1]);
   if ((infile = fopen(filename,"rb")) == NULL)
   {
      printf("**********Prepare cannot open %s", filename);
      return;
   }
   strcpy(inname,argv[1]);
   strcpy(filename,argv[2]);
```

```
strcat(&filename,".XMT");
outfile = fopen(filename,"wb");
fputs("IBM Mainframe XFER PREPARE File\n",outfile);
fprintf(outfile,"%s\n",inname);

i = 2;
rest = 0;
countin = 0;
countout = 0;

while ((c = getc(infile)) != EOF)
{
    countin++;
    i = (i+1) % 3;
    /* First prepare the output character
        Shift right 2(n+1) places (n = 0,1,2)
        and add the rest of the preceeding character. */
    outc = c >> (2*(i + 1));
    outc = rest + outc;
    putc(outc+32,outfile);
    countout++;
    /* Next, prepare the rest of the character
        Shift left 4-2n places (n = 0,1,2)
        and then mask off everything past 6 bits. */
    rest = (c << (4 - 2*i)) & 0x3f;
    /* Output the rest of the third character as a six
        bit character in its own right. */
    if (i == 2)
    {
        putc(rest+32,outfile);
        countout++;
        rest = 0;
    }
}
/* Handle the last character. */
if (i != 2)
{
    putc(rest+32,outfile);
    countout++;
}

/* Put in an end marker character */
outc = 100;
putc(outc,outfile);

printf("\nBytes input:   %ld",countin);
printf("\nBytes output:  %ld",countout);
fclose(infile);
```

```
    fclose(outfile);
}
```

Appendix B

UNPACK Routine Listing

```
#include <stdio.h>
#define FNL 80   /* maximum file name length */

main(argc,argv)
   int argc;
   char *argv[];

/* Author:  Norman R. Lyons
             Naval Postgraduate School

   Date:    January 1986

   Version: 1.0

   Calling Sequence:  UNPACK infile

   The UNPACK routine takes a file of six bit words
   generated by the PREPARE routine and converts it
   into one of eight bit words.  The infile name is
   assumed to be infile.xmt, and the first line of
   the file is:
   IBM Mainframe XFER PREPARE File
   The second line of the file is the name of the
   original file processed by the PREPARE routine.
   This name is used for the output file. */

{
   FILE *infile, *outfile, *fopen();

   char filename[FNL],text[80];
   int c,i,outc,rest;
   long int countin,countout;
   strcpy(filename,argv[1]);
   strcat(&filename,".XMT");
   if ((infile = fopen(filename,"rb")) == NULL)
   {
      printf("**********Unpack cannot open %s", filename);
      return;
```

```
}
fgets(text,80,infile);
i = strncmp("IBM Mainframe XFER PREPARE File",text,31);
if (i != 0)
{
   printf("**********%s is not a PREPARE file.",filename);
   return;
}
fgets(filename,80,infile);
printf("\n\nThe output file name will be:");
printf("\n\n     %s",filename);
printf("\nIs this acceptable to you?");
printf("\n1 = Yes, anything else = No\n");
if ((i = getchar()) != '1') return;
/* Delete the newline character */
strncpy(filename,filename,strlen(filename)-1);
outfile = fopen(filename,"wb");

countin = 0;
countout = 0;
if ((outc = getc(infile)) != EOF)
{
   outc = outc - 32;
   outc = (outc << 2) & 0xff;
   countin++;
}
i = 2;
rest = 0;

while ((c = getc(infile)) != 100)
/* A 100 (the character d) is used as a stop character. */
{
   c = c - 32;
   countin++;
   i = (i+1) % 3;
   /* First prepare the output character
      by adding the rest from the next character. */
   rest = c >> (4 - 2*i);
   outc = rest + outc;
   putc(outc,outfile);
   countout++;
   /* Now, prepare the next output character
      Shift c left 4+2n places (n = 0,1,2)
      and then mask off everything past 8 bits. */
   if (i != 2) outc = (c << (4 + 2*i)) & 0xff;
   /* If there is no remaining data in the next word, read in a
      new character. */
   else if ((outc = getc(infile)) != EOF)
```

```
        {
            outc = outc - 32;
            outc = (outc << 2) & 0xff;
            ++countin;
        }

    }
    printf("\nBytes input:   %ld",countin);
    printf("\nBytes output:  %ld",countout);
    fclose(infile);
    fclose(outfile);

}
```

Appendix C

ARC Documentation

ARC

File Archive Utility
Version 4.5

This file describes the ARC file utility, version 4.4, which was created by System Enhancement Associates on 25 October 1985.

ARC is the copyrighted property of System Enhancement Associates. You are granted a limited license to use ARC, and to copy it and distribute it, provided that the following conditions are met:

1. No fee may be charged for such copying and distribution.

2. ARC may ONLY be distributed in its original, unmodified state.

Any voluntary contributions for the use of this program will be appreciated, and should be sent to:

System Enhancement Associates
21 New Street
Wayne, NJ 07470

You may not use this product in a commercial environment or a governmental organization without paying a license fee of 35. Site licenses and commercial distribution licenses are available. A program disk and printed documentation are available for 50.

A word about user supported software:

The user supported software concept (usually referred to as "freeware") is an attempt to provide software at low cost. The cost of offering a new product by conventional channels is staggering, and hence dissuades many independant authors and small companies from developing and promoting their ideas. User supported software is an attempt to develop a new marketing channel, where products can be introduced at low cost.

If user supported software works, then everyone will benefit. The user will benefit by receiving quality products at low cost, and by being able to "test drive" software thoroughly before purchasing it. The author benefits by being able to enter the commercial software arena without first needing large sources of venture capital.

But it can only work with your support. We're not just talking about ARC here, but about all user supported software. If you find that you are still using a program after a couple of weeks, then pretty obviously it is worth something to you, and you should send in a contribution.

And now, back to ARC:

ARC is used to create and maintain file archives. An archive is a group of files collected together into one file in such a way that the individual files may be recovered intact.

ARC is different from other archive and library utilities in that it automatically compresses the files being archived, so that the resulting archive takes up a minimum amount of space.

When ARC is used to add a file to an archive it analyzes the file to determine which of four storage methods will result in the greatest savings. These four methods are:

1. No compression; the file is stored intact.

2. Repeated-character compression; repeated sequences of the same byte value are collapsed into a three-byte code sequence.

3. Huffman squeezing; the file is compressed into variable length bit strings, similar to the method used by the SQ programs.

4. Lempel-Zev compression; the file is stored as a series of twelve bit codes which represent character strings, and which are created "on the fly".

Note that since one of the four methods involves no compression at all, the resulting archive entry will never be larger than the original file.

USING ARC

ARC is invoked with a command of the following format:

```
ARC <x> <arcname> [<template> . . .]
```

Where:

<x> is an ARC command letter (see below), in either upper or lower case.

<arcname> is the name of the archive to act on, with or without an extension. If no extension is supplied, then ".ARC" is assumed. The archive name may include path and drive specifiers.

<template> is one or more file name templates. The "wildcard" characters "*" and "?" may be used. A file name template may only include a path or drive specifier if you are adding a file to an archive.

If ARC is invoked with no arguments (by typing "ARC", and pressing "enter"), then a brief command summary is displayed.

Following is a brief summary of the available ARC commands:

```
a   = add files to archive
m   = move files to archive
u   = update files in archive
f   = freshen files in archive
d   = delete files from archive
x,e = extract files from archive
r   = run files from archive
p   = copy files from archive to stdout
l   = list files in archive
v   = verbose listing of files in archive
t   = test archive integrity
c   = convert entry to new packing method
b   = retain backup copy of archive
s   = suppress compression (store only)
w   = suppress warning messages
n   = suppress notes and comments
```

These commands are explained in more detail below.

ARC COMMANDS

ADDING FILES

Files are added to an archive using the "A" (Add), "M" (Move), "U" (Update), or "F" (Freshen) commands. Add always adds the file. Move differs from Add in that the source file is deleted once it has been added to the archive.

Update differs from Add in that the file is only added if it is not already in the archive, or if it is newer that the corresponding entry in the archive.

Freshen is similar to Update, except that new files are not added to the archive; only files already in the archive are updated.

For example, if you wish to add a file named "TEST.DAT" to an archive named "MY.ARC", you would use a command of the form:

```
ARC a my test.dat
```

If you wanted to add all files with a ".C" extension, and all files named "STUFF" to an archive named "JUNK.ARC", you could type:

```
ARC a junk *.c stuff.*
```

If you wanted to move all files in your current directory into an archive named "SUM.ARC", you could use a command of the form:

```
ARC m sum *.*
```

If you have an archive named "TEXT.ARC", and you wanted to add to it all of your files with an extension of ".TXT" which have been created or changed since they were last archived, then you would type:

```
ARC u text *.txt
```

If you have a bunch of files in your current directory, with backup copies being stored in an archive named "SAFE.ARC", then if you wanted to make sure that every file in the archive is the latest version of that file, you would type:

```
ARC f safe
```

A word about Update and Freshen: These are similar in that they look at the date and time of last change on the file, and only add it if the file has been changed since it was last archived. They differ in that Update will add new files, while Freshen will not.

In other words, Update looks for the files on disk, and adds them if they are new or have changed, while Freshen looks in the archive, and tries to update the files which are already there.

Archive entries are always maintained in alphabetic order. Archive entries may not have duplicate names. If you add a file to an archive that already contains a file by that name, then the existing entry in the archive is replaced. Also, the archive itself and its backup will not be added.

You may also add a file which is in a directory other than your current directory. For example, it is perfectly legal to type:

```
ARC a junk c:\dustbin\stuff
```

The A, U, and M commands are the ONLY commands which allow you to give a drive or path. Also, you cannot add two files with the same name. In other words, if you have a file named "C:\DUSTBIN\STUFF.TXT" and another file named "C:\BUCKET\STUFF.TXT", then typing:

```
arc a junk c:\dustbin\*.* c:\bucket\*.*
```

will not work.

ARC will never add an archive to itself, nor will it add the temporary copy or a backup copy of the archive.

DELETING FILES

Archive entries are deleted with the "D" (Delete) command. For example, if you had an archive named "JUNK.ARC", and you wished to delete all entries in it with a filename extension of ".C", you could type:

```
ARC d junk *.c
```

EXTRACTING FILES

Archive entries are extracted with the "E" (Extract) and "X" (eXtract) commands. For example, if you had an archive named "JUNK.ARC", and you wanted all files in it with an extension of ".TXT" or ".DOC" to be recreated on your disk, you could type:

```
ARC x junk *.txt *.doc
```

If you wanted to extract all of the files in an archive named "JUNK.ARC", you could simply type:

```
ARC x junk
```

Whatever method of file compression was used in storing the files is reversed, and uncompressed copies are created in the current directory.

RUNNING FILES

Archive entries may be run without being extracted by use of the "R" (Run) command. For example, if you had an archive named "JUNK.ARC" which contained a file named "LEMON.COM", which you wished to run, you could type:

```
ARC r junk lemon.com
```

You can run any file from an archive which has an extension of ".COM", ".EXE", or ".BAT". You cannot run interpretive BASIC programs from an archive, nor can you give arguments to a program you are running from an archive.

In practice, the file to be run is extracted, run, and then deleted. All in all, this is a fairly useless command. Does anyone actually use it? If not, we'd like to get rid of it.

PRINTING FILES

Archive entries may be examined with the "P" (Print) command. This works the same as the Extract command, except that the files are not created on disk. Instead, the contents of the files are written to standard output. For example, if you wanted to see the contents of every ".TXT" file in an archive named "JUNK.ARC", but didn't want them saved on disk, you could type:

```
ARC p junk *.txt
```

If you wanted them to be printed on your printer instead of on your screen, you could type:

```
ARC p junk *.txt >prn
```

LISTING ARCHIVE ENTRIES

You can obtain a list of the contents of an archive by using the "L" (List) command or the "V" (Verbose list) command. For example, to see what is in an archive named "JUNK.ARC", you could type:

```
ARC l junk
```

If you are only interested in files with an extension of ".DOC", then you could type:

```
ARC l junk *.doc
```

ARC prints a short listing of an archive's contents like this:

```
Name          Length    Date
============  ========  =========
ALPHA.TXT         6784  16 May 85
BRAVO.TXT         2432  16 May 85
COCO.TXT           256  16 May 85
        ====  ========
Total      3      9472
```

"Name" is simply the name of the file.

"Length" is the unpacked file length. In other words, it is the number of bytes of disk space which the file would take up if it were extracted.

"Date" is the date on which the file had last been modified, as of the time when it was added to the archive.

"Total" is pretty obvious, I think.

ARC prints a verbose listing of an archive's contents like this:

```
Name          Length    Stowage   SF   Size now  Date       Time    CRC
============  ========  ========  ==== ========  =========  ======  ====
ALPHA.TXT         6784  Squeezed  35%      4413  16 May 85  11:53a  8708
BRAVO.TXT         2432  Squeezed  41%      1438  16 May 85  11:53a  5BD6
COCO.TXT           256   Packed    5%       244  16 May 85  11:53a  3AFB
        ====  ========            ==== ========
Total      3      9472            27%      6095
```

"Name", "Length", and "Date" are the same as for a short listing.

"Stowage" is the compression method used. The following compression methods are currently employed:

--	No compression.
Packed	Runs of repeated byte values are collapsed.
Squeezed	Huffman squeeze technique employed.
Crunched	Lempel-Zev compression technique employed.

"SF" is the stowage factor. In other words, it is the percentage of the file length which was saved by compression. The "total" stowage factor is the stowage factor for the archive as a whole, not counting archive overhead.

"Size now" is the number of bytes the file is occupying while in the archive.

"Time" is the time of last modification, and is associated with the date of last modification.

"CRC" is the CRC check value which has been stored with the file. Another CRC value will be calculated when the file is extracted or tested to ensure data integrity. There is no especially good reason for displaying this value.

TESTING AN ARCHIVE

The integrity of an archive may be tested by use of the "T" (Test) command. This checks to make sure that all of the file headers are properly placed, and that all of the files are in good shape.

This can be very useful for critical archives, where data integrity must be assured. When an archive is tested, all of the entries in the archive are unpacked (without saving them anywhere) so that a CRC check value may be calculated and compared with the recorded CRC value.

For example, if you just received an archive named "JUNK.ARC" over a phone line, and you want to make sure that you received it properly, you could type:

```
ARC t junk
```

It defeats the purpose of the T command to combine it with N or W.

CONVERTING AN ARCHIVE

The "C" (Convert) command is used to convert an archive entry to take advantage of newer compression techniques. For example, if you had an archive named "JUNK.ARC", and you wanted to make sure that all files with an extension of ".DOC" were encoded using the very latest methods, you could type:

```
ARC c junk *.doc
```

Or if you wanted to convert every file in the archive, you could type:

```
ARC c junk
```

SUPRESSING COMPRESSION

The "S" (Supress compression) option can be combined with any command that updates archive entries. These include Add, Move, Update, Freshen, and Convert. The effect of the S option is to prevent any compression techniques from being employed. This is intended to allow you to add a few files at a time to an archive quickly, and then later convert the archive to compress everything at once.

For example, over the course of a day you might give each of the following commands:

```
ARC as junk *.txt
ARC as junk *.mac
ARC as junk *.doc
```

At the end of the day, when you have finished adding things to the archive, you could have all of the archive entries compressed at once by typing:

```
ARC c junk
```

You could also "decompress" the archive by typing:

```
ARC cs junk
```

though I can't imagine why you'd want to.

BACKUP RETENTION

When ARC changes an archive (during an Add, Move, Update, Freshen, Delete, or Convert) it creates a new archive with the same name, but with an extension of ".   ". For example, if you add a file to an archive named STUFF.ARC, then ARC will create a new archive named STUFF.   . ARC will read from your existing archive and write out the new archive with any changes to the ".   " copy.

Normally when ARC is finished it deletes the original and renames the new archive to the original name (ie. STUFF.ARC goes away, and STUFF.   becomes the new STUFF.ARC). Among other things, this means that if anything goes wrong and ARC is unable to finish, then your original archive will still be intact.

In some circumstances you may wish to retain the original version of the archive as a backup copy. You can do this easily by using the Backup option. Add the letter "B" to your command, and ARC will rename your original archive to have an extension of ".BAK" instead of deleting it.

In other words, if you wanted to add "WASTE.TXT" to an archive named "JUNK.ARC", but wanted to keep a backup copy, then you would type:

```
ARC ab junk waste.txt
```

Your original archive would become "JUNK.BAK", while "JUNK.ARC" would contain the new "WASTE.TXT" file.

If you keep a backup of an archive which already has a backup, then the older backup copy is deleted.

MESSAGE SUPPRESION

ARC prints two types of messages, warnings and comments.

Warnings are messages about suspected error conditions, such as when a file to be extracted already exists, or when an extracted file fails the CRC error check. Warnings may be suppressed by use of the "W" (Warn) command. You should use this command sparingly. In fact, you should probably not use this command at all.

Comments (or notes) are informative messages, such as naming each file as it is added to the archive. Comments and notes may be suppressed by use of the "N" (Note) command.

For example, suppose you extracted all files with an extension of ".BAS" from an archive named "JUNK.ARC" Then, after making some

changes which you decide not to keep, you decide that you want to extract them all again, but you don't want to be asked to confirm every one. In this case, you could type:

```
ARC xw junk *.bas
```

Or, if you are going to add a hundred files with an extension of ".MSG" to an archive named "TRASH.ARC", and you don't want ARC to list them as it adds them, you could type:

```
ARC an trash *.msg
```

Or, if you want to extract the entire contents of an archive named "JUNK.ARC", and you don't want to hear anything, then type:

```
ARC xnw junk
```

RAMDISK SUPPORT

If you have a ramdisk, or other high-speed storage, then you can speed up ARC somewhat by telling it to put its temporary files on the ramdisk. You do this by setting the ARCTEMP environment string with the MS-DOS SET command. For example, if drive B: is your ramdisk, then you would type:

```
set ARCTEMP=B:
```

Refer to the MS-DOS manual for more details about the SET command. You need only set the ARCTEMP string once, and ARC will use it from then on until you reboot or change its value.

SPECIAL NOTES

The function used to calculate the CRC check value in previous versions has been found to be in error. It has been replaced in version 3.0 with a proper function. ARC will still read archives created with earlier versions of ARC, but it will report a warning that the CRC value is in error. All archives created prior to version 3.0 should be unpacked and repacked with the latest version of ARC.

Transmitting a file with XMODEM protocol rounds the size up to the next multiple of 128 bytes, adding garbage to the end of the file. This used to confuse ARC, causing it to think that the end of the archive was invalidly formatted. This has been corrected in version 3.03.

Older archives may still be read, but ARC may report them to be improperly formatted. All files can be extracted, and no data is lost. In addition, ARC will automatically correct the problem when it is encountered.

VERSION NUMBERS

There seems to be some confusion about our version numbering scheme. All of our version numbers are given as a number with two decimal places.

The units indicate a major revision, such as adding a new packing algorythm.

The first decimal place (tenths) indicates a minor revision that is not essential, but which may be desired.

The second decimal place (hundredths) indicates a trivial revision that will probably only be desired by specific individuals or by diehard "latest version" fanatics.

ARC also displays its date and time of last edit. A change of the date and time without a corresponding change in version number indicates a truly trivial change, such as fixing a spelling error.

To sum up: If the units change, then you should get the newer version. If the tenths change, then you may want to get the newer version. If anything else changes, then you probably shouldn't bother. This is reflected by our own habit of referring to "version 4.3" instead of "version 4.31".

CHANGES IN VERSION 4

ARC is adding another data compression technique in this version. We have been looking for some technique that could improve on Huffman squeezing in at least a few cases. So far, Lempel-Zev compression seems to be fulfilling our fondest hopes, often acheiving compression rates as much as 20% better than squeezing, and sometimes even better. Huffman squeezing depends on some bytes being more "popular" than others, taking the file as a whole. Lempel-Zev compression is instead looking for long strings of bytes which are repeated at various points (such as an end of line followed by spaces for indentation). Lempel-Zev compression is therefor looking for repetition at a more "macro" level, often acheiving impressive packing rates.

Alas, nothing ever comes free. This gain in storage efficiency comes at the price of processor time. ARC version 4.0 will usually take about twice as long to add a file to an archive as version 3.1 did. We intend to work on improving this in the future, but it will always be slower since it must now work much harder to determine the best packing method.

Fortunatly, file extraction is only slightly slower, to the point where it will probably go unnoticed.

In the typical case a file is added to an archive once and then extracted many times, so the increased time for an update should more than pay for itself in increased disk space and reduced file transmission time. As usual, ARC version 4.0 is completely upward compatible. That is, it can deal properly with any archive created by any earlier version of ARC. It is NOT reverse compatible. Archives created by ARC 4.0 will generally not be usable by earlier versions of ARC.

CHANGES IN VERSION 4.1

Version 4.1 does not contain any major changes from version 4.0. Lempel-Zev coding has been improved somewhat by performing non-repeat compression on the data before it is coded (as was already done with Huffman squeezing). This has the two fold advantage of (a) reducing to some extent the amount of data to be encoded, and (b) increasing the time it takes for the string table to fill up. Performance gains are small, but noticable.

The primary changes are in internal organization. ARC is now much "cleaner" inside. In addition to the esthetic benefits to the author, this should make life easier for the hackers out there. There is also a slight, but not noticable, improvement in overall speed when doing an update.

Version 4.1 is still fully upward compatible. But regretfully, it is again not downward compatible. Version 4.1 can handle any existing archive, but creates archives which older versions (including 4.0) cannot unpack.

CHANGES IN VERSION 4.3

Version 4.3 adds the much-demanded feature of using pathnames when adding files to an archive. For obscure technical reasons, files being

extracted still go in the current directory on the current drive. Pathnames are also not supported for any of the other commands, because it would make no sense.

Version 4.3 is also using a slightly different approach when adding a file to an archive. The end result is twofold:

1. Slightly more disk space is required on the drive containing the archive. This should only be noticeable to those creating very large archives on a floppy based system.

2. A 30% reduction in packing time has been achieved in most cases. This should be noticeable to everyone.

As always, version 4.3 is still fully upwards compatible, and is backwards compatible as far as version 4.1.

CHANGES IN VERSION 4.4

The temporary file introduced in version 4.3 occasionally caused problems for people who had not added a FILES= statement to their CONFIG.SYS file. This has now been corrected. Also, support of the ARCTEMP environment string was added to allow placing of the temporary file on a ramdisk.

A bug was reported in the Run command, which has been fixed. From the extreme time required before the bug was reported, it is deduced that the Run command is probably the least used feature of ARC.

The Update command was changed. It is no longer a straight synonym for Add. Instead, Update now only adds a file if it is newer than the version already in the archive, as shown by the MS-DOS date/time stamp.

CHANGES IN VERSION 4.5

The Convert command was not making use of ramdisk support. Now it is.

The Freshen command was added. Our first choice for a name was Refresh, but we already had a Run command. Assuming that you have an archive which already contains everything you want in it (for software distribution, perhaps), then Freshen would be used to update the archive. It was pointed out to us that ARC already knows what is in the

archive, so it should be able to look on disk for newer versions. Now it can.

The Suppress compression option was added by popular demand. It allows files to be added quickly to an archive, since the files are not analyzed or compressed, but merely stored straight. The intent is to allow users to build an archive "in pieces", and then compress all of the entries at once with the Convert command. The conversion is much faster if you take advantage of ramdisk support.

A minor bug was detected in our handling of date/time stamps which occasionally resulted in stamping an archive with the wrong date and time. This has been corrected.

PROGRAM HISTORY AND CREDITS

In its short life thus far, ARC has astounded us with its popularity. We first wrote it in March of 1985 because we wanted an archive utility that used a distributive directory approach, since this has certain advantages over the more popular central directory approach. We added automatic squeezing in version 2 at the prompting of a friend. In version 2.1 we added the code to test for the best compression method. Now (in October of 1985) we find that our humble little program has spread across the country, and seems to have become a new institution.

We are thankful for the support and appreciation we have received. We hope that you find this program of use.

If we have acheived greatness, it is because we have stood upon the shoulders of giants. Nothing is created as a thing unto itself, and ARC is no exception. Therefore, we would like to give credit to the following people, without whose efforts ARC could not exist:

Brian W. Kernighan and P. J. Plauger, whose book "Software Tools" provided many of the ideas behind the distributive directory approach used by ARC.

Dick Greenlaw, who wrote the public domain SQ and USQ programs, in which the Huffman squeezing algorithm was first developed.

Robert J. Beilstein, who adapted SQ and USQ to Computer Innovations C86 (the language we use), thus providing us with important parts of our squeezing logic.

Kent Williams, who graciously allowed us to use his LZWCOM and LZWUNC programs as a basis for our Lempel-Zev compression logic.

David Schwaderer, whose article in the April 1985 issue of PC Tech Journal provided us with the logic for calculating the CRC 16 bit polynomial.

And many, many others whom we could not identify.

Distribution List

Recipient	No. Copies
Director of Research Code 014 Naval Postgraduate School Monterey, CA 93943	1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library Code 1424 Naval Postgraduate School Monterey, CA 93943	2
Professor Norman R. Lyons Code 54LB Naval Postgraduate School Monterey, CA 93943	10
Mr. Edward Cain NCS/PP 8th Street and South Courthouse Road Arlington, VA 22204	2
Mr. Ken Boheim NCS/PP 8th Street and South Courthouse Road Arlington, VA 22204	1
Dr. Bruce Barrow NCS/PP 8th Street and South Courthouse Road Arlington, VA 22204	1
Mr. Norman Douglas NCS/EP 8th Street and South Courthouse Road Arlington, VA 22204	1
COL William Schooler NCS/EP 8th Street and South Courthouse Road Arlington, VA 22204	1

LTC Tom Cindric 1
JDSSC
8th Street and South Courthouse Road
Arlington, VA 22204

Professor M. G. Sovereign 1
Code 55ZO
Naval Postgraduate School
Monterey, CA 93943

Professor James R. Yee 1
Code 55YE
Naval Postgraduate School
Monterey, CA 93943

Professor Carl R. Jones 1
Code 54JS
Naval Postgraduate School
Monterey, CA 93943

Professor Jack W. La Patra 1
Code 54LP
Naval Postgraduate School
Monterey, CA 93943

Professor Norman F. Schneidewind 1
Code 54SS
Naval Postgraduate School
Monterey, CA 93943

Professor Tung Bui 1
Code 54BD
Naval Postgraduate School
Monterey, CA 93943

Professor Dan Dolk 1
Code 54DK
Naval Postgraduate School
Monterey, CA 93943

Professor Taracad Sivasankaran 1
Code 54SE
Naval Postgraduate School
Monterey, CA 93943